LA-UR- 01-1723

Approved for public release;
distribution is unlimited.

Title: TRex: Interactive Texture Based Volume Rendering for Extremely Large Datasets

Author(s): Joe Kniss, The University of Utah
Patrick McCormick, LANL, CCS-1
Allen McPherson, LANL, CCS-1
James Ahrens, LANL, CCS-1
Jamie Painter, TurboLabs
Alan Keahey, LANL, CCS-1
Charles Hansen, The University of Utah

Submitted to: IEEE Computer Graphics and Applications Special Issue on Large Data Visualization (Aug/Sept. 2001 issue)

Los Alamos
NATIONAL LABORATORY

TRex: Interactive Texture Based Volume Rendering for Extremely Large Datasets

Joe Kniss† Patrick McCormick‡ Allen McPherson‡ James Ahrens‡ Jamie Painter‡ Alan Keahey‡
Charles Hansen†

†Scientific Computing and Imaging Institute
School of Computing, University of Utah

‡Advanced Computing Laboratory
Los Alamos National Laboratory

Abstract

Many of today's scientific simulations are capable of producing terabytes to petabytes of data. Visualization plays a critical role in understanding and analyzing the results of these simulations. Hardware accelerated direct volume rendering has proven to be an excellent visualization modality for both scientific and medical data sets. Current graphics hardware implementations limit the size of interactive datasets to sizes that are orders of magnitude smaller than these datasets.

We present a scalable system which takes advantage of parallel graphics hardware, software based compositing, and high performance I/O. The goals of our application are to provide near interactive display rates for terabyte sized, time-varying, datasets and allow moderately sized datasets to be visualized in virtual environments. We also present a novel set of direct manipulation widgets for interacting with, and querying, the visualization.

1 Introduction

Visualization is an integral part of scientific computation and simulation. State of the art simulations of physical systems can generate terabytes to petabytes of data where a single time step can contain more than a gigabyte of data per variable. Sizes of some datasets are projected to increase at a rate close to Moore's law. The key to understanding this data, is the ability to visualize the global and local relationships of data elements. Direct volume rendering has proven to be an excellent method for examining these properties. It allows each data element to contribute to the final image and provides the ability to query not only the spatial relationship of data elements, but their quantitative relationships as well. Hardware accelerated volume rendering is an approach that allows users to achieve interactive display rates for reasonably sized datasets. The size of interactive datasets is a function of the hardware's available texture memory and fill rate. Current high end hardware implementations place an upper bound on dataset sizes of around 256MB. In this paper, we present a scalable, pipelined approach for rendering datasets which are larger than the limitations of a single graphics card. We do this by taking advantage of multiple hardware rendering units and parallel software compositing.

The goals of TRex, our system for interactive volume rendering of large datasets, are to provide near interactive display rates for terabyte sized, time-varying, datasets and provide a low latency platform for volume visualization in immersive environments. We consider 5 frames per second (fps) to be near interactive rates for normal viewing environments, and immersive environments to have a lower bound frame rate of 10 fps. While this is significantly below most virtual environment update rates, we have found that the user can successfully investigate extremely large datasets at this rate. Using TRex for virtual reality environments requires very low latency, around 50ms per frame, or 100ms per view update or stereo pair. To achieve lower latency renderings, we either render smaller portions of the volume on more graphics pipes or subsample the volume so that a graphics pipe renders fewer samples per frame. We present previous volume rendering work in the next section. We then provide an overview of our TRex renderer. This is followed by a brief discussion on multipipe compositing issues. We then describe the enhancements we provided for immersive environments. We close with our current efforts in extending this system.

2 Previous Work

Current volume rendering methods can be grouped into three major categories. The first are those that lend themselves to parallel software implementations. These include ray casting, shear warp, and splatting. Ray casting [6] is essentially a special case of the ray tracing method. Rays are cast from an eye point through the view plane and intersected with volume elements. Samples taken along the ray are typically trilinearly interpolated and composited in a front to back order. Shear warp [4] takes advantage of precomputed coordinate axis aligned slices through the volume and replaces the more expensive trilinear interpolation with bilinear in these slice planes. This method generates arbitrary view points by orthographically compositing the sheared slices along a major axis. This intermediate view produces a sheared version of the volume which then undergoes a 2D warping transformation in image space to generate the final image. Shear warp is considered one of the fastest software volume rendering methods, but can suffer from artifacts. It also requires three copies of the volume to remain in memory, one copy for each major axis. Splatting [9] is projection based method where each voxel is generalized into a contribution extent, typically achieved by convolving the voxel with a gaussian, which essentially eliminates the need for interpolation since the samples are the voxels themselves. This method also typically composites the splats in a front to back order.

The second grouping includes methods which are implemented on single graphics adapters. While at some level the hardware may take advantage of parallelism, the volume rendering is performed on a single graphics unit. Using texture mapping hardware, one can take two dimensional, axis aligned slices which take advantage of bilinear texture mapping hardware [1]. These slices are alpha-blended to form the final image. Two-dimensional texture methods also suffer from the same sampling artifacts as the software shear warp implementation and require three copies of the data sliced along the major axes. Three dimensional texture based methods [10] take advantage of trilinear interpolation in hardware. The voxels are mapped onto polygons aligned with the view directions using trilinear interpolation based upon three dimensional

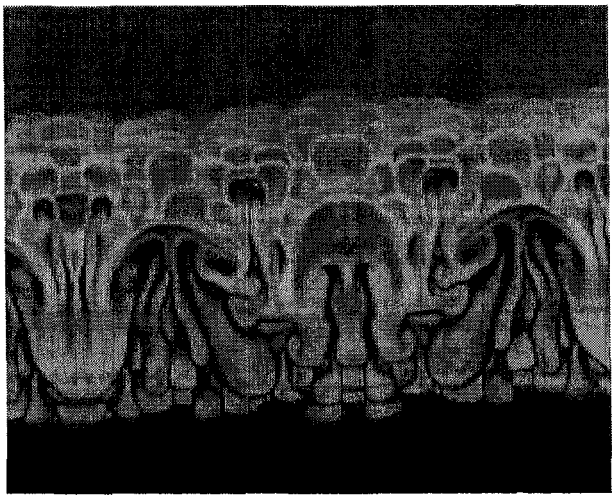


Figure 1: Time-Varying Raleigh Taylor Fluid Instability, RAGE dataset (1024^3).

texture mapping hardware. Lamar introduced a method of polygon slice construction based on concentric shells [5] which better approximates the ray casting method above. The VolumePro volume rendering board provides a hardware implementation of the ray casting method and can achieve high frame rates for 512^3 volumes [7].

The third grouping includes methods which utilize hardware graphics units in parallel. Previous work in this area includes the Minnesota batch mode parallel hardware volume renderer, implementation by Paul Woodward [11]. This volume renderer was implemented for use on an Origin 2000. It was not designed to provide interactive visualization, rather users create key frames which the application uses to produce animations. Volumizer, a proprietary API from SGI for hardware volume rendering, also supports parallel volume rendering. The frame latency of this API, however, increases linearly as more graphics pipes are added. Both implementations composite in hardware. This requires each partial image to be downloaded and composited in the user interface's frame buffer. Downloading multiple images to graphics hardware can take considerable time, thus limiting the interactivity of these implementations. Furthermore, parallel rendering using Volumizer pipelines the compositing sequentially along the different graphics pipes. This leads to limited scaling with an n frame latency where n is the number of graphics pipes.

3 TRex System Overview

The implementation presented in this paper is a hybrid parallel software and hardware volume renderer. It was designed to meet target performance for near interactive display for large scale time-varying scientific datasets. Specifically, the system was designed to render full resolution, 1024^3, time-varying data, such as the RAGE dataset in Figure 1, at nearly 5 fps on an 4-pipe Onyx-2 with IR-3 graphics hardware. The limiting factor is the high performance I/O as described in section 3.2.1. Rendering is not the bottleneck as demonstrated by achieving over 5 fps on an 4-pipe Onyx-2 with a static 1024^3 volume. For immersive environments, we achieve 10 fps using stereo pairs, albeit with reduced resolution as described in section 5. The primary difference from previous parallel hardware volume rendering work is the volume renderer's interactivity on extremely large datasets. This is achieved by a design that utilizes all available hardware components: streaming time-varying datasets from a carefully designed, very high-performance I/O system, rendering with graphics pipes and compositing the results with processors. We have found that interactively visualizing datasets is a critical first step in the analysis process, allowing users to rapidly gain a detailed understanding of their results. Since the hardware components can work independently, the parallel volume rendering process can be pipelined as shown in Figure 2. With overlapped stages, this pipeline allows us to achieve an overall performance which closely matches the performance of an individual graphics pipe. We review the details of the TRex pipeline in the following sections.

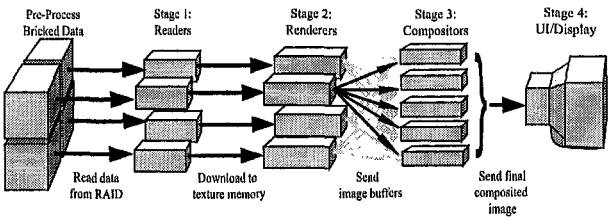


Figure 2: The TRex Pipeline

3.1 Pre-processing

Our implementation requires an off-line preprocessing step in which the data is quantized from its native data type, commonly floating point, to either 8 bit or 12 bit unsigned integer data. The data is then split into subvolumes with sizes matching the available texture memory on each graphics pipe. Note that most graphics hardware implementations currently require texture dimensions to be a power of two. The original volume may need to be super sampled or padded to match this power of two requirement. It is advisable to perform these operations on the original floating point data prior to bricking to avoid quantization errors and artifacts at subvolume interfaces. Interface artifacts are caused by boundary conditions in super sampling schemes.

3.2 The TRex Pipeline

TRex's rendering pipeline has four stages. Each stage is a multithreaded process capable of executing simultaneously with the other stages. A stage consists of two main parts: an event manager which handles communication, and a functional part which implements the task(s) of the thread. For a volume partitioned into N subvolumes, TRex will create N readers and N renderers. Note that it is not necessary for the number of compositing threads to equal the number of subvolumes. The ideal number of compositor threads is a function of both image size and the number of images to be composited. A user interface thread is responsible for displaying the final composited image and sending user event messages to the other stages and the threads for supporting Immersive TRex and direct manipulation widgets. We discuss each of the pipeline stages in the following sections.

3.2.1 Stage 1: Subvolume Reader

The first stage of the pipeline involves reading a time step from disk. TRex creates a separate reader thread for each of the subvolumes in a time step. Provided the data resides on a well striped RAID and direct I/O is available, the subvolumes can be read from disk

in parallel at approximately 140MB/sec. [1] Unfortunately this data rate is not fast enough to sustain our desired throughput of 5 fps for 1024^3 time-varying datasets but does provide a parallel approach to I/O that will work on most Silicon Graphics systems. For large time-varying datasets, a higher performance I/O design is required.

A method to achieve such high performance I/O is to build a file system that is customized for streaming volume data directly into system memory and then into texture memory. On the SGI Origin 2000 it is possible to do this by first co-locating the I/O controllers and graphics pipes, such that they share a common physical memory within the NUMA architecture. This configuration avoids the overhead associated with routing data through the system's interconnection network, thus minimizing data transfer latency. In order to maximize performance, our goal for I/O is to match the approximate 300MB/sec texture download rate of the InfiniteReality pipes. For 16 pipes operating in parallel this is equivalent to a sustained rate of approximately 5GB/sec. These rates require the use of a striped file system built using 64 dual fiber channel controllers and 2, 304 individual disks. This is achieved by placing 4 fiber channel controllers per pipe, with each fiber channel controller capable of 70MB/sec. This gives us a rate of 4×70MB/sec$=$ 280MB/sec for each pipe. Assuming we achieve the best case performance, we are limited by this 280MB/sec rate of the I/O system. In addition we have also discovered that it is necessary to store subvolumes in contiguous blocks on disk to achieve these data rates. This can be done by using SGI's real time filesystem (RTFS). Our initial benchmarks have placed this configuration capable of approximately 4GB/sec. We are continuing our efforts to reach the desired 5GB/sec rate.

3.2.2 Stage 2: Render

The second stage of the pipeline renders subvolumes in parallel. Each graphics pipe is managed by a separate rendering thread. These threads are initially responsible for creating OpenGL rendering windows. A renderer initializes multiple image buffers for simultaneous rendering with the other stages since the compositing stage and UI stage both rely on the image buffers as well. Raw data buffers, for the Reader Stage, are created by the rendering threads so that the memory will reside near the graphics units it will be rendered on; provided that the rendering thread has been placed on a processor near the unit. The number of raw data buffers is dependent on the amount of time step buffering desired plus one for the simultaneous reading of data and downloading to texture memory.

Renderers receive a render message from the user interface. This message includes information about the current frame's rotation, translation, scale, and sample rate. The OpenGL model-view matrix is set for the frame then geometry and volume data are rendered. Each renderer supports a simplistic scene graph which orders geometric primitives and subvolumes (if a pipe renders more than one). Our volume rendering approach uses 3D textures with either view aligned slicing [10] or an approximation to Lamar's concentric shells method [5]. The final image's color and alpha values are read from the frame buffer and stored in a buffer. Finally, the renderer sends a message to the compositors that a new image is available, along with a pointer to the image buffer, and the subvolume's distance from the eye point.

A texture lookup table encodes the transfer function. It assigns color and alpha values to the scalar texture elements. The user makes changes to the transfer function by manipulating control points in color and alpha space as illustrated in Figure 3. We have extended the transfer function control to allow the user to select a boundary distance function based on Kindlmann's semi-automatic transfer function generation [3]. The user can then select the appropriate portions of this automatic transfer function by manipulating control points in the alpha band (see Figure 4). If the transfer function or sample rate has changed since the last frame, the transfer function is updated and redownloaded to the graphics hardware prior to rendering a subvolume.

[1] Direct I/O avoids kernel interrupt overhead and is a feature available on SGI systems.

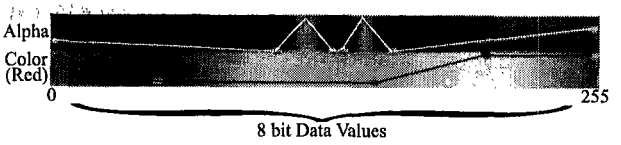


Figure 3: Texture Lookup Table. Note: the Alpha Band (top) has been multiplied by the Color Band (bottom) to show the resulting alpha weighted colors.

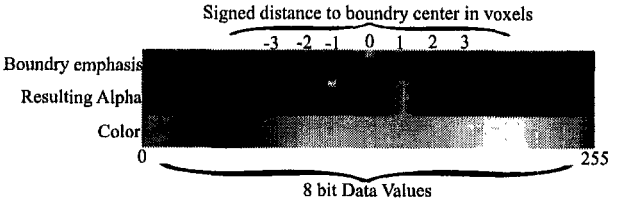


Figure 4: Texture Lookup Table with the Semi-Automatic generation of alpha mappings. The top band allows the user to select data values based on their distance from an ideal boundary detected in the volume. The middle band shows the generated alpha mapping multiplied by the color band.

3.2.3 Stage 3: Compositor

Compositing begins once a completion message is received from each of the N renderers. The message includes a pointer to the renderer's shared memory image buffer and the subvolume's distance from the eye point. A compositing thread is responsible for compositing N images across a horizontal stripe of the final image. Composite order is determined by comparing the locations of the subvolumes and compositing back to front. As shown in Figure 5 when image A is composited over image B, the resulting image resides in A's buffer. This eliminates the need for additional memory in the compositing stage. The first composite thread waits for the other compositors to finish before sending a message to the UI that a new image is ready for display.

The decision to use software compositing over hardware compositing was made because the task is embarrassingly parallel and

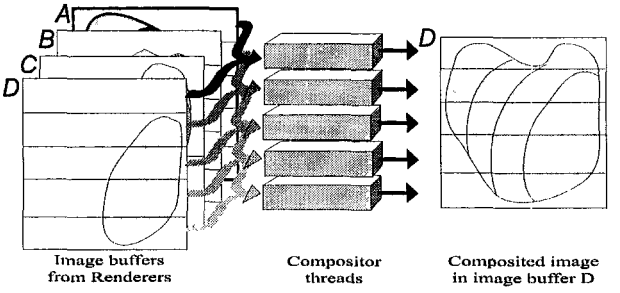


Figure 5: Stage 3, compositing, detail. In this example 4 image buffers from stage 2 are composited as D over C over B over A. Image buffer D would then be downloaded to the UI's frame buffer in Stage 4.

the cost of downloading N images to the graphics hardware is prohibitively time consuming. Also, the graphics hardware is the critical resource in this system. By utilizing available CPUs to composite the partial results, we can achieve better scaling than a graphics hardware approach. Employing the available CPUs also allows us to overlap compositing with the rendering of the next frame resulting in only a one frame latency rather than the multiple frame latency imposed by the Volumizer and Minnesota hardware based compositing systems.

3.2.4 Stage 4: User Interface

The user interface thread is responsible for managing the input from the user and sending messages that trigger other stages of the pipeline. If the user changes a viewing parameter such as rotation, scale, translation, or the transfer function, the UI sends a request to the renderers along with the new view parameters for the frame. When a message is received indicating that a new frame is available, the UI downloads the raw image data from the shared memory image buffer directly to the display's frame buffer.

In addition, the user has access to a quality parameter that adjusts the number of samples through the volume. This parameter is also set automatically. When the user is in an interaction state such as a pending rotation or translation, the sample rate is set lower to increase the frames per second. Once the interaction state is completed, i.e. the user releases the mouse click, the quality parameter is set higher and the volume is rendered at a higher sample rate allowing automatic progressive refinement. When the window size is changed, the UI will send a resize request to the renderers. The slave display's window size will be changed as well as the image buffers.

4 Discussion

Applications rendering transparent objects, from back to front, generate new color values by using Equations 1 and 2 [8].

$$c_{out} = a_{source} \times c_{source} + (1 - a_{source}) \times c_{target} \quad (1)$$

$$a_{out} = a_{source} + (1 - a_{source}) \times a_{target} \quad (2)$$

Where a_{target} and c_{target} are the alpha and color values currently in the frame buffer. a_{source} and c_{source} are the incoming alpha and color values. a_{out} and c_{out} are the new alpha and color values to be written to the frame buffer.

Standard hardware implementations only allow the setting of one function which applies to both color and alpha channels. Since the equations for color and alpha are clearly different, we cannot simply apply color and alpha bending with Equation 1. Doing so would cause errors in the accumulated alpha value. Equations 3 and 4 demonstrate what happens when alpha compositing is treated the same as color compositing.

$$c_{out} = a_{source} \times c_{source} + (1 - a_{source}) \times c_{target} \quad (3)$$

$$a_{out} = a_{source} \times a_{source} + (1 - a_{source}) \times a_{target} \quad (4)$$

Notice that $alpha_{source}$ is squared and then added to the complemented $alpha_{source}$ times $alpha_{target}$. This contrasts with Equation 2 and the error can not be easily corrected.

The solution is to pre-multiply the color values in the texture lookup table by their corresponding alpha values. The resulting Equations 5 and 6 match Equations 1 and 2 respectively, since ca_i expands to $c_i * a_i$.

$$c_{out} = ca_{source} + (1 - a_{source}) \times c_{target} \quad (5)$$

$$a_{out} = a_{source} + (1 - a_{source}) \times a_{target} \quad (6)$$

This correction is only necessary when the alpha values are used at some latter time, such as compositing. For display on a single graphics pipe, the accumulated alpha value is not important, since only the incoming fragment's alpha value is used in computing the color value, i.e. the alpha value in the frame buffer is never used for computing color.

TRex allows the use of a variable sampling rate by allowing an arbitrary number of slices through the volume. In this situation, it is important to properly scale the alpha values of incoming slices so that the overall look of the volume is maintained regardless of the sample rate. The relationship between the sample rate and scaled alpha values is not linear. Equation 7 approximates this relationship.

$$alpha_{new} = 1 - (1 - alpha_{old})^{\frac{sr_{old}}{sr_{new}}} \quad (7)$$

Where sr_{old} is the sample rate used with $alpha_{old}$ and sr_{new} is the new sample rate used with $alpha_{new}$.

It is also important to note that this non-linear scaling of alpha values is only critical for alpha values near or less than 0.2. The scaling of alpha values as they approach 1 have a near linear behavior. In practice the computation required to scale alpha values based on Equation 4 is expensive. This, however, is compensated for by the fact that the texture look up tables are relatively small in size, 256 or 4096 elements, when compared to the size of the volume data.

The use of polygonal objects such as direct manipulation widgets and isosurfaces in conjunction with volume data requires us to take steps to insure that the geometry is composited with the volume correctly. A scene with geometry and volume composited correctly allows geometry to appear embedded in the volume. We currently limit the type of geometric objects to those that are fully opaque. Geometry is rendered first with depth test and depth write. Next the volume data is rendered from back to front with depth test only. This allows volume data to be rendered over geometric data but not behind. One difficulty of compositing subvolumes with geometric data is handling polygons that reside in two or more subvolumes or only partially in a subvolume. This can be solved by clipping geometry to planes corresponding to the faces of the subvolume which border other subvolumes. This requires at most six clipping planes for a subvolume which is completely surrounded by other subvolumes.

TRex takes advantage of several platform specific optimizations. Our benchmarks on the IR graphics subsystem revealed that unsigned shorts were best for frame buffer reads and writes. Topology of the platform is also a concern especially in NUMA platforms like the Origin 2000. Data transfer latency can be significantly reduced by placing processes and their memory nearest the I/O devices that they use. This configuration avoids the overhead associated with routing data through the system's interconnection network. A significant speed up was realized by placing the renderers and incoming data buffers near the IR pipe which they manage for the same reasons.

5 Immersive TRex

Using TRex in an immersive environment adds another level of complexity to the rendering pipeline. First, a stereo pair must be generated for each new viewpoint. This requires twice the fill rate as a monocular viewpoint. Achieving target frame rates requires that we decrease the number of samples that each graphics pipe must process per frame. One solution is to increase the number of graphics pipes, rendering threads, and compositors. This requires rebricking the dataset so that smaller subvolumes are rendered on

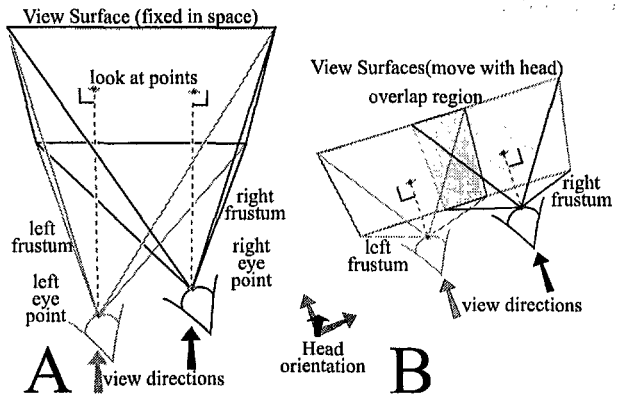


Figure 6: View Point construction for Semi-Immersive Environments (A), and Fully-Immersive Environments (B) for an arbitrary head orientation. Notice: View directions are parallel and perpendicularly intersect the view surface. The overlap region in B is variable on many head mounted displays

each pipe. Reducing the overall size of the dataset via subsampling is another option if additional graphics units are not available. Subsampling, however, has the side effect of blurring fine details. The number of compositer threads in either case needs to be increased to match the lower latency of the rendering stage.

Second, tracking devices are essential for creating an immersive environment. Typically a separate daemon process communicates with the tracking device and reports position and orientation to a shared memory arena. A TRex VR session will create an additional thread for monitoring and reporting head and hand/device data to the UI. Since multiple tracking devices are available, such as *Polhemus* Fast Track(TM) and *Ascension* Flock of Birds(TM), and interaction devices may vary, the VR thread is responsible for mapping events from the current tracking and interaction devices to a set of events which the TRex UI understands.

New viewpoints are generated using parallel viewing directions with asymmetrical, or sheared, frustums (see Figure 6). Head orientation in semi-immersive environments such as the Responsive Workbench(TM *Fakespace*) is essentially disregarded. This is because we can treat eyes as points, and assume the portal to the virtual space (view surface) is fixed in the real space. View direction for semi-immersive environments is determined by the line from the eye point in real space which perpendicularly intersects the view surface plane. For fully immersive environments, such as those achieved with $n-Visions$ Datavisor(TM) head mounted displays, eyes are still treated as points but head orientation is required to specify the virtual portal. View direction for fully immersive environments is specified by head orientation, and eyes are assumed to be looking forward.

View aligned slicing causes artifacts when the volume is close to the user and rendered with perspective projection. Lamar's spherical shell slicing reduces this problem by assuring that the volume is rendered with differential slices which are perpendicular to the line from the center of projection to the volume element being rendered. Our approach uses an adaptive tessellation of the spherical shell. While coarse tessellations can cause artifacts, fine tessellations can cause significant latency in the rendering. We allow the user to select a tessellation which is appropriate for the visualization.

6 TRex Widgets:

Direct manipulation widgets [2] can improve the quality and productivity of interactive visualization. Widgets also allow the user to have a uniform experience when using either the desktop and an immersive environment.

Our widget sets were created using the Brown University widget paradigm. The widgets are object oriented, extendible entities which maintain state similar to a discrete finite automaton. They are based on simple parts such as spheres, bars, and sliders. Complex widgets are constructed from these sub-parts. Each sub-part represents some functionality of the widget. For instance, the bars which make up the boundary of a frame widget when selected would translate the whole frame, the spheres which bracket the corners would scale the frame, sliders attached to the bars would alter some scalar value associated with the functionality of the widget.

To facilitate parallel rendering, a typical widget is broken into N+1 lightweight threads, where N is the number of subvolumes in the session. A parent thread is responsible for handling events from the UI and communicating with the child threads. The N child threads are responsible for rendering the widget and performing subvolume specific tasks. Each child thread is associated with a subvolume and is clipped to the half planes corresponding to the faces of the subvolume which border other subvolumes. We have developed three custom widgets for use with TRex.

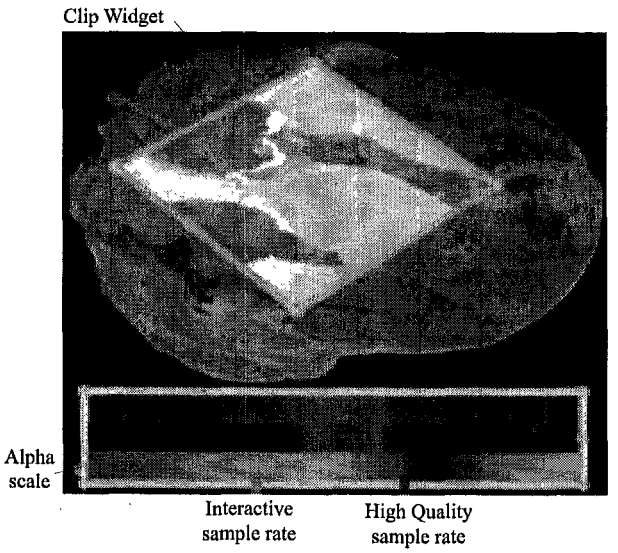
The color map widget places the transfer function in the scene with the volume being rendered. This provides a tighter coupling between the actual data values and their representation as a color value in the image. The color map widget is composed of three bands, one for color to data value mapping, one for opacity to data value mapping, and one for the semi-automatic generation of opacity mappings [3]. This widget also includes sliders, as can be seen in Figure 7(a), for manipulating the high quality sample rate, interactive sample rate, and opacity scaling.

We developed a data probe widget to allow the user to query the visualization for local quantitative information. This widget can be used to point to a region of the volume and automatically query the original data for the values at that location. This widget is particularly useful for studying data from physical simulations where the actual value at a location is of interest. Figure 7(c) shows a dataset employing the data probe widget.

The internal structure of volumetric data is often obscured by other portions of the volume. One method for revealing hidden information is to use clipping planes to remove the regions which occlude. For this purpose we developed a widget which allows the user to position and orient an arbitrary clipping plane in the scene. Because of the amorphous quality of some volume renderings, it is necessary to map a slice to the clipping plane with a different transfer function to make the clipped boundary apparent. One useful mapping is a simple data value to gray scale and a linear alpha ramp from low to high (see Figure 7(b)). This sort of mapping is particularly useful for radiology datasets where users are more accustomed to viewing slices, rather than the whole volume.

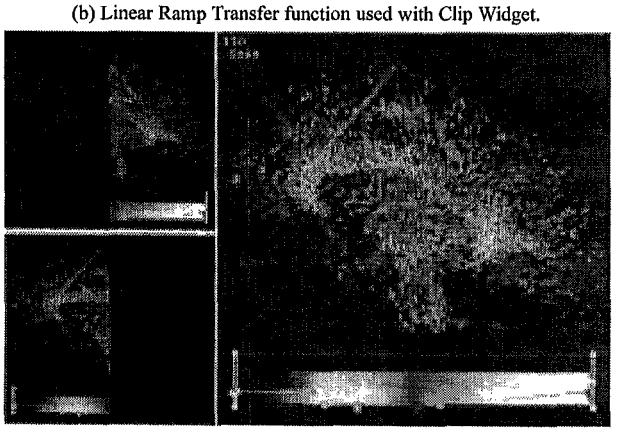
7 Conclusions and Future work

Hardware volume rendering is a highly effective interactive visualization modality. Unfortunately, it imposes limits on the size of volumetric datasets which can be rendered with adequate update rates. We have presented a scalable solution for the near interactive visualization of dataset time steps which are potentially an order of magnitude larger than the capabilities of a modern graphics card alone. Our implementation is also flexible enough to support advanced interaction tools and serve as a platform for future volume rendering and visualization research. The results of our research



(a) Clip and Color Map widgets, Visualization is a MRI of a sheep heart (512^3) with cut-away showing atrium, ventricle, and valve.

(b) Linear Ramp Transfer function used with Clip Widget.



(c) Diffusion Tensor MRI of a human brain 512^3 on 2 graphics pipes

Figure 7: TRex Widgets, (a) demonstrates the Color Map and Clip Widgets, (c) demonstrates the Data Probe widget and shows intermediate renderings to the left.

are currently being integrated in a production quality application for use by scientists at Los Alamos National Laboratory.

With the recent performance gains in the commodity graphics card market, we are currently investigating PC clusters as a replacement for the Origin 2000 system presented in this paper. While we have successfully managed to get TRex running on a PC cluster, this task presents several significant challenges. The limitations present in both the internal PC architecture and interconnection networks will make it difficult to maintain our near interactive rendering rates. There are also several areas in which our current algorithms need to be modified so they operate efficiently in a distributed memory environment. In addition, the new functionality available in the rasterization hardware of recent commodity graphics cards offer a number of shading options at a per pixel level. We intend to implement parallel, diffuse shaded volumes, as well as explore alternative shading methods.

Currently, TRex only supports opaque geometry which must be downloaded and rendered on each graphics pipe. We intend to extend TRex to render both opaque and transparent geometry embedded within the volume data in a parallel, load balanced, fashion.

Finally, we are enhancing TRex's VR capabilities with intuitive interaction devices and new widgets. We will be exploring optimizations to TRex's pipeline, such as predictive tracking, which take advantage of the known inter-frame latency. Direct manipulation widgets have proven to be an indispensable tool for interactive visualization and immersive environments. We are developing a suite of widgets to perform various operations on volumetric data such as classification, segmentation, annotation, editing, multiple channel and vector volume visualization. In addition to their virtual world representation, widgets can have a physical representation associated with the interaction devices employed. To this end, we intend to develop custom interaction devices derived from common hand held and desktop devices specifically for this type of visualization. We are also considering the addition of other visualization modalities such as haptic feedback and auralizaion.

8 Acknowledgments

This work was supported in part by grants from the DOE ASCI VIEWS program and the DOE AVTC. The RAGE dataset was used courtesy of Robert Weaver, LANL.

References

[1] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *ACM Symposium On Volume Visualization*, 1994.

[2] D. Brookshire Conner, Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, Robert C. Zeleznik, and Andries van Dam. Three-Dimensional Widgets. In *Proceedings of the 1992 Symposium on Interacive 3D Graphics*, pages 183–188, 1992.

[3] Gordon Kindlmann and James W. Durkin. Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering. In *ACM Symposium On Volume Visualization*, pages 79–86, 1998.

[4] Philip Lacroute and Marc Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transform. In *ACM Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 451–458, July 1994.

[5] Eric LaMar, Bernd Hamann, and Kenneth I. Joy. Multiresolution Techniques for Interactive Texture-Based Volume Visualization. In *Proceedings Visualization '99*, pages 355–361. IEEE, October 1999.

[6] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, 8(5):29–37, 1988.

[7] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The VolumePro Real-Time Ray-Casting System . In *ACM Computer Graphics (SIGGRAPH '99 Proceedings)*, pages 251–260, August 1999.

[8] Thomas Porter and Tom Duff. Compositing Digital Images. In *ACM Computer Graphics (SIGGRAPH '84 Proceedings)*, pages 253–259, July 1984.

[9] Lee Alan Westover. *Splatting: A Parallel, Feed-Forward Volume Rendering Algorithm.* PhD thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, 1991.

[10] Orion Wilson, Allen Van Gelder, and Jane Wilhelms. Direct Volume Rendering via 3D Textures. Technical Report UCSC-CRL-94-19, University of California at Santa Cruz, June 1994.

[11] P. R. Woodward. Interactive Scientific Visualization of Fluid Flow. *IEEE Computer*, pages 13–25, Oct. 1993.